# Why We should learn programming?

# Why should Tester learn programming?

Fault [Error/Bug]

- How can a person's **fault** finding be effective, if he/she doesn't have enough understanding of these programs?

# Why should Tester learn programming?

Reviewers

- Industry looks at testers, as reviewers for the programs, who can suggest how to improve quality of programs on a continuous basis. Is this really possible without knowledge of programming?

# Why should Tester learn programming?

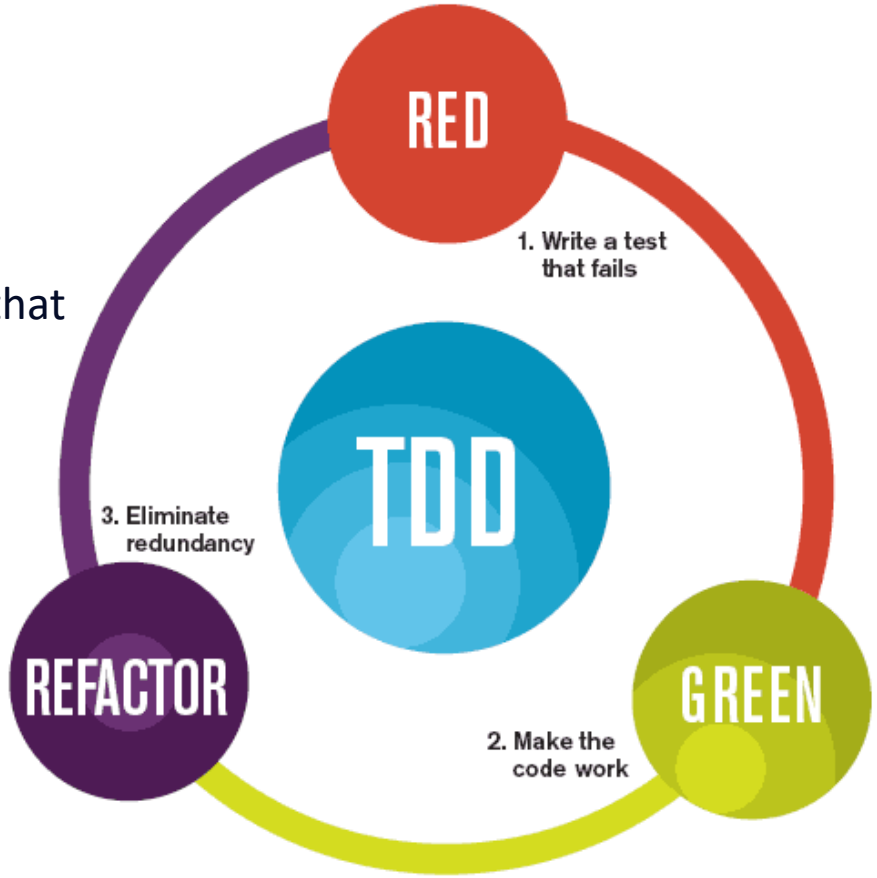
Automation

- Test automation is developing a script or a program for the purpose of programmatic testing of software


# Why should Tester learn programming?

## TDD Code Katas

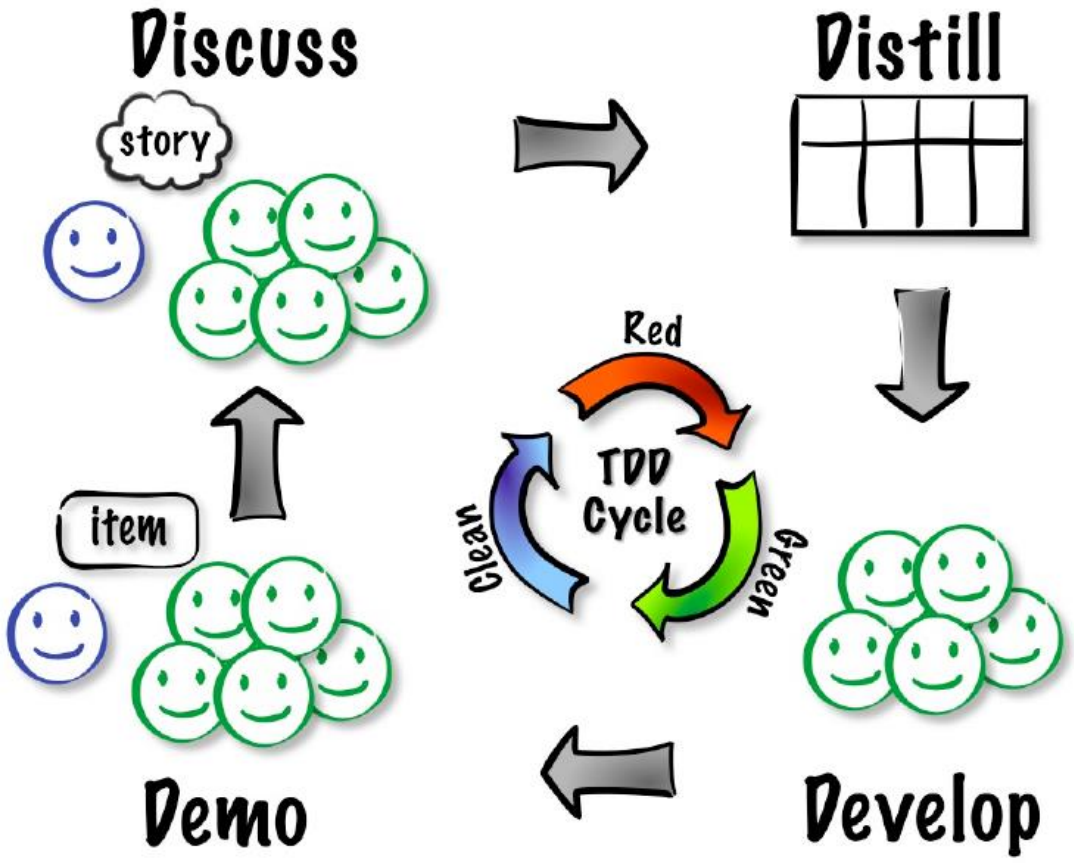
Test-driven development (TDD) is a software development practice that emphasizes writing tests before writing the actual code.




The mantra of Test-Driven Development (TDD) is "red, green, refactor."

# Why should Tester learn programming?

Acceptance TDD



# Java Programming


Java™ Education
and Technology Services

Presented By
Dr. Eman Hesham, DBA.

Invest In Yourself ,
Develop Your Career

A None-stop learner, Instructor, Coach, Mentor, Consultant ,SME, Researcher who is always self-motivated , Passionate to learn

- got DBA in modern teaching methodologies
- got her MSc in the data security field
- is Certified Application Security Engineer - CASE - EC-Council
- is Oracle Certified Web Component Developer
- is Certified Agile Scrum Master
- is Teaching Specialist in AI & DS -EPITA
- got ITI- 9-month diploma, intake 26, Java Department
- is Subject Matter Expert (SME ) for online course creation & production at Mahar-Tech
  - Java Course with 28,000 + online learner
- is Training Consultant for training and development for different companies
- 18+ years of experience of teaching and development
  - ITI students( 9month, 3 month , AI program and Egyptian universities )
  - Africa
  - Ministry of defense and Ministry of finance
  - Multinational companies.
- Coding for kids and STEM Coach , Mentor and Judge for many years
- Applies Agile in training & online tech technologies.
- participated in the organization and preparation of MCIT, ITI and JETS events( JETS Launching Seminar and Workshop, MDW)


Dr. Eman Hesham , DBA.

MCIT- ITI - JETS

Executive Manager

ehesham@iti.gov.eg

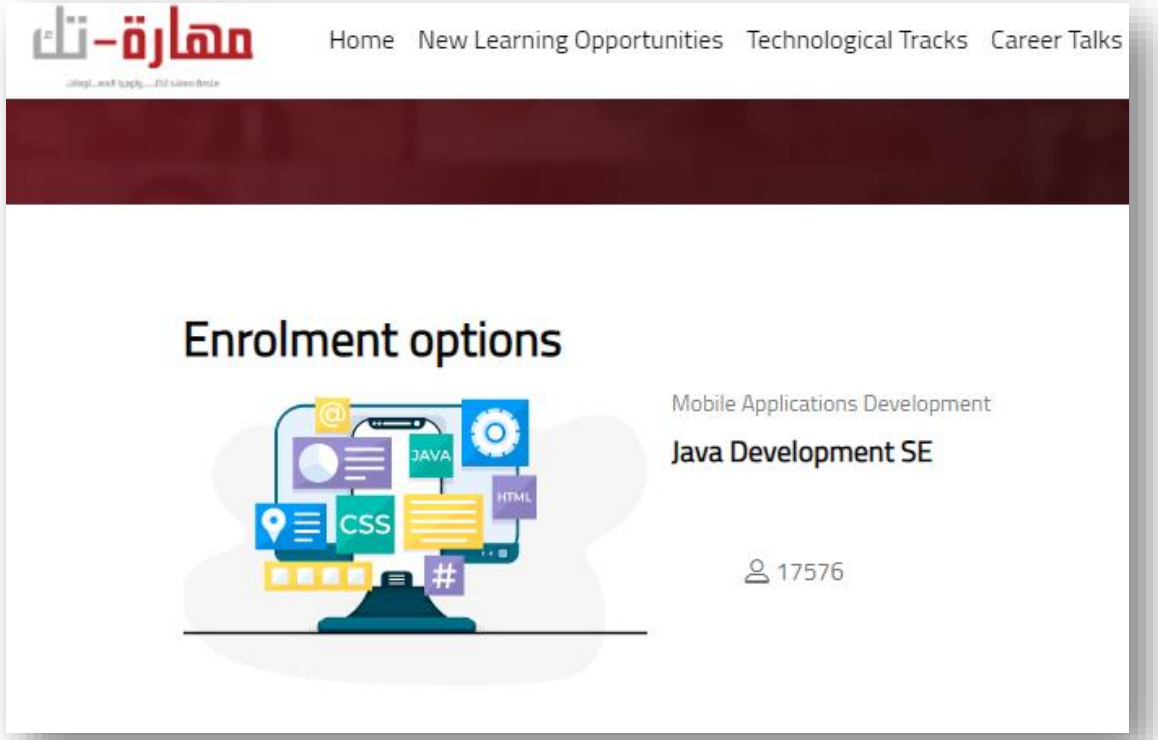
Hesham.eman@gmail.com

Linkedin: eman-hesham-dba-m-sc-550a981b/

# Online Java development Course

- General
  - Announcements
- Introduction to OOP Using Java
- OOP II and Applet
- Controlling Program Flow
- Modifiers-Access Specifiers Essential Java Classes
- Using Arrays & Strings
- Data Types & Operators
- Interfaces
- Multi-Threading
- Inner Classes
- Event Handling

- https://maharatech.gov.eg/enrol/index.php?id=12



# Course Outline

Java™ Education and Technology Services

## Day1

- Introduction to Java
- Basic Java Concepts

## Day2

- Data Types & Operators
- using Arrays & Strings
- Controlling Program Flow
- Modifiers and Access Specifiers

## Day3

- Simple GUI
- Essential Java Classes
- Java Exception

## Day4

- Interfaces
- Multi-Threading

## Day5

- Inner class
- Event Handling

# Agenda


I. Introduction to Java

1. Java History
2. Java Principles and Features

II. Basic Java Concepts

1. Java and OOP
2. Java Structure

III. Installing JDK and Apache NetBeans

# Agenda


I. Introduction to Java

1. Java History
2. Java Principles and Features

II. Basic Java Concepts

1. Java and OOP
2. Java Structure

III. Installing JDK and Apache NetBeans

# Brief History of Java

- Java was developed by Sun Microsystems in may 1995.
- The Idea was to create a language for controlling any hardware, but it was too advanced.
- A team - that was called the Green Team - was assembled and lead by James Gosling.
- Platform and OS Independent Language.
- Oracle Corporation is the current owner of the official implementation of the Java SE platform
- Free License; cost of development is brought to a minimum.

# Brief History of Java

- From mobile phones to handheld devices, games and navigation systems to e-business solutions, Java is everywhere!
- Java can be used to create:
  - Desktop Applications,
  - Web Applications,
  - Enterprise Applications,
  - Mobile Applications,
  - Smart Card Applications.
  - Embedded Applications

# Agenda


I. Introduction to Java

1. Java History
2. Java Principles and Features

II. Basic Java Concepts

1. Java and OOP
2. Java Structure

III. Installing JDK and Apache NetBeans

# Java Principles

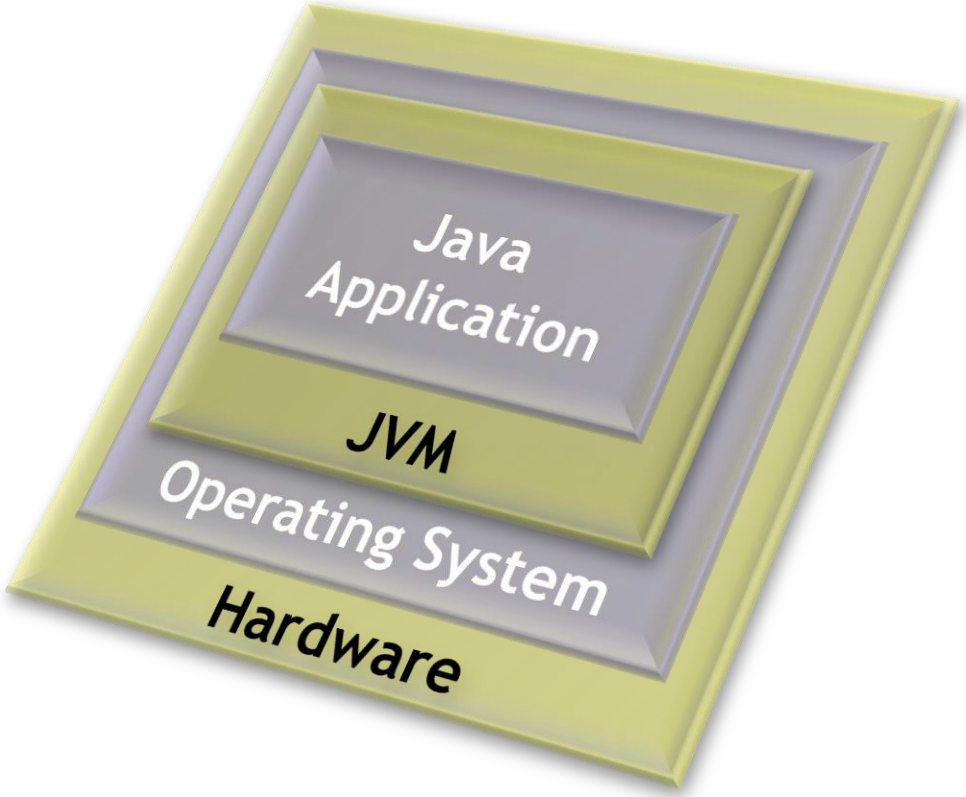
- Primary goals in the design of the Java programming language:
  - Simple
  - Object oriented
  - Distributed
  - Multithreaded
  - Dynamic
  - Portable
  - High performance
  - Robust
  - Secure

# Java Features

- Java is easy to learn!
  - Syntax of C++
  - Dynamic Memory Management (Garbage Collection)
  - No pointers

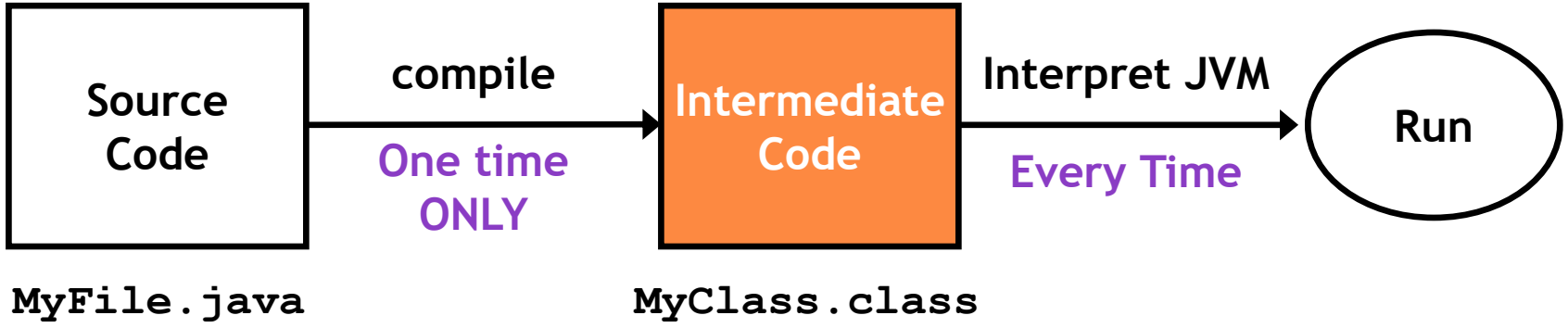
# Java Features

- Machine and Platform Independent (Architecture Neutral)



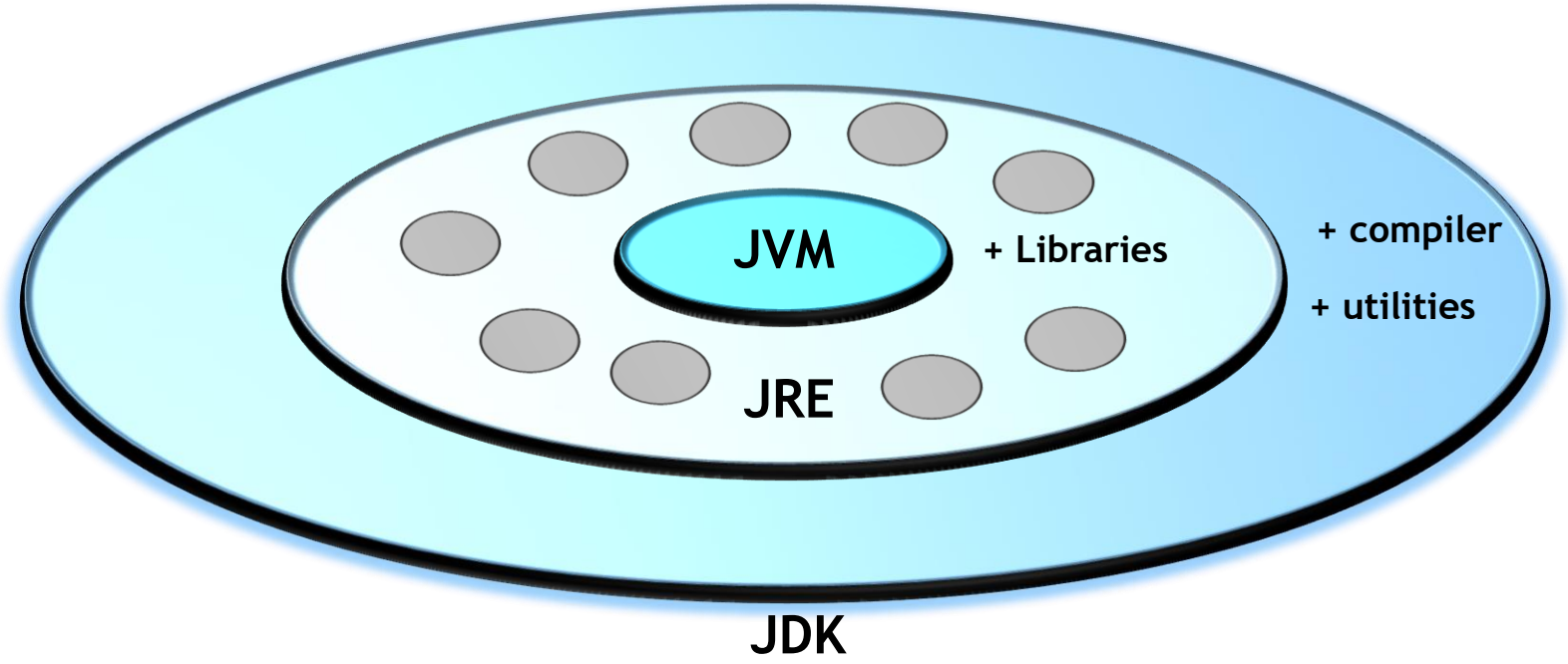
# Java Features

- Java is both, compiled and interpreted



# Java Features

- Java depends on dynamic linking of libraries



# Java Features

- Java is fully Object Oriented
  - Made up of Classes.
  - No multiple Inheritance.
- Java is a multithreaded language
  - You can create programs that run multiple threads of execution in parallel.
    - Ex: GUI thread, Event Handling thread, GC thread
- Java is networked
  - Predefined classes are available to simplify network programming through Sockets(TCP-UDP)

# Agenda


I. Introduction to Java
1. Java History
2. Java Principles and Features

II. Basic Java Concepts
1. Java and OOP
2. Java Structure

III. Installing JDK and Apache NetBeans

# Agenda

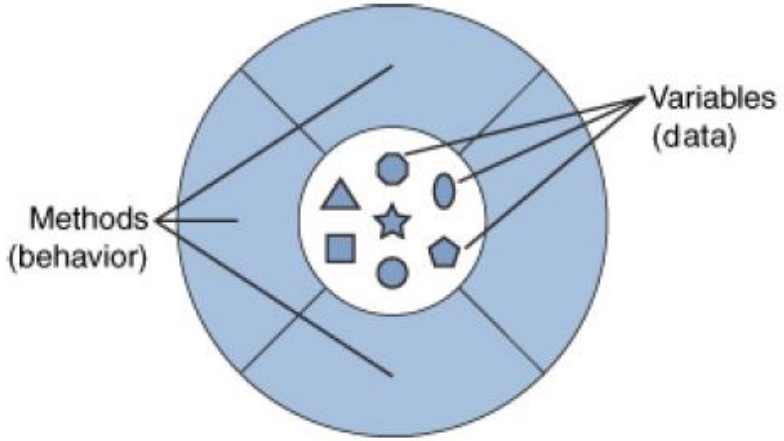

I. Introduction to Java
1. Java History
2. Java Principles and Features

II. Basic Java Concepts
1. Java and OOP
2. Java Structure

III. Installing JDK and Apache NetBeans

# Introduction to OOP - Object

- **What is an Object?**
  - An object is a software bundle of variables and related methods.
- Object consist of:
  - Data (object's Attributes)
  - Behavior (object's methods)



# Introduction to OOP - Class

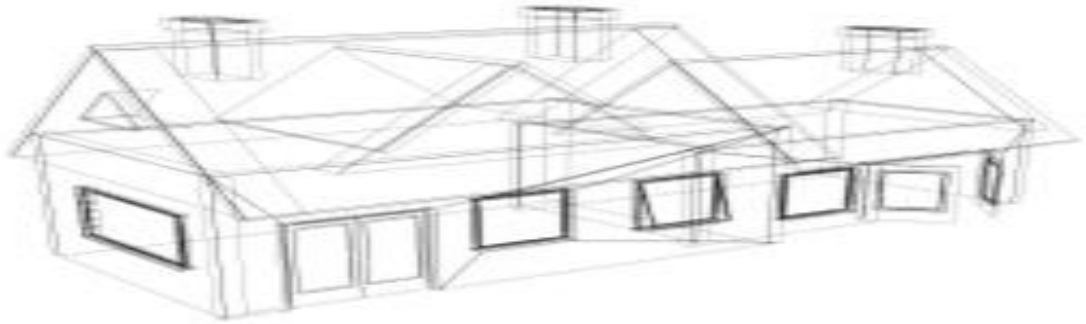
Java™ Education and Technology Services

- **What is a Class?**
  - A class is a blueprint of objects.
  - A class is an object factory.
  - A class is the template to create the object.
  - A class is a user defined datatype
- **Object:**
  - An object is an instance of a class.
  - The property values of an object instance is different from the ones of other object instances of a same class
  - Object instances of the same class share the same behavior (methods).

# Introduction to OOP – Object & Class

- ***Class*** reflects concepts.
- ***Object*** reflects instances that embody those concepts.

***class***



***objects***

# How to create a class?

- To define a class, we write:

```
<access-modifier>* class <name>
{
        <attributeDeclaration>*
        <constructorDeclaration>*
        <methodDeclaration>*
}
```

- Example:

```
class StudentRecord {
        //we'll add more code here later
}
```

# Coding Guidelines

- Think of an appropriate name for your class.
  - Don't use XYZ or any random names.
- Class names starts with a CAPITAL letter.
  - not a requirement it is a convention

# Declaring Properties (Attributes)

- declare a certain attribute for our class, we write,

```
<access-modifier>* <type> <name> [= <default_value>];
```

- Example:

```
class StudentRecord {
            // Instance variables
            public String       name;
            public String       address;
            private int         age    =      15;
            /*we'll add more code here later */
}
```

# Declaring Properties (Attributes)

- **Access modifiers:**

1. **Public attributes:**
   - The access availability inside or outside the class.
2. **Private attributes:**
   - The access availability within the class only.
3. **Protected attributes:**
   - The access availability within the class and its subclasses only.

# Declaring Methods

- declare a certain method for our class, we write,

```
<modifier>* <Return type> <name> ([<Param Type> <Param
Name>]*)
{
        <Statement>*
}
```

- Example:

```
class StudentRecord {
        private String name;
        public String getName(){ return name; }
        public void setName(String str){ name=str; }
        public static String getSchool(){..........}
}
```

# Declaring Methods

- The following are characteristics of methods:
  - It can return one or no values
  - It may accept as many parameters it needs or no parameter at all.
    - Parameters are also called arguments.
  - After the method has finished execution, it goes back to the method that called it.
  - Method names should start with a small letter.
  - Method names should be verbs.

# Declaring Properties (Methods)

- **Access modifiers:**

1. **Public method:**
   - The access availability inside or outside the class.
2. **Private method:**
   - The access availability within the class only.
3. **Protected attributes:**
   - The access availability within the class and its subclasses only.

4. **Static method:**
   - Methods that can be invoked without instantiating a class.
   - To call a static method, just type,

```
Classname.staticMethodName(params);
```

# Big Example

```
class Student{
    String firstName,lastName;
    int age;
    double mathScore;
    double scienceScore;
 int getAge(){ return age; }
    void setAge(int g){ age=g; }

 public static String getSchool(){//return school name}

 public double average(){
      double avg=0;
      avg=(mathScore+scienceScore)/2;
      return avg;
    }
}
```

# Create Object Instance

- To create an object instance of a class,
  - we use the `new` operator.
- For example,
  - if you want to create an instance of the class Student, we write the following code,

```
Student s1 = new Student();
```

- The new operator
  - Allocates a memory for that object and returns a reference of that memory location to you.
  - When you create an object, you actually invoke the class' constructor.

# Accessing members of class

- To access members of class:

```
class Test {
    void testMethod(){
        Student s1 = new Student();
        s1.setAge(10);
        double d;
        d = s1.average();
        String s = Student.getSchool();
    }
}
```

# First Java Application

```
class HelloWorld
{
  public static void main(String[] args)
  {
   System.out.println("Hello Java");
  }
}
```

File name: `hello.java`

# First Java Application cont'd

- The `main()` method:
  - Must return void.
  - Must be static.
    - because it is the first method that is called by the Interpreter (`HelloWorld.main(..)`) even before any object is created.
  - Must be public to be directly accessible.
  - It accepts an array of strings as parameter.
    - This is useful when the operating system passes any command arguments from the prompt to the application.

# System.out.println("Hello");

- **out** is a static reference that has been created in class **System**.
- **out** refers to an object of class **PrintStream**. It is a ready-made stream that is attached to the standard output (i.e. the screen).

```
public class System
{
   public static PrintStream out;
   ..............
}
```

```
public class PrintStream
{
   public void print(String str)
   {...........}

   public void println(String str)
   {...........}
}
```

# Agenda


I. Introduction to Java
   1. Java History
   2. Java Principles and Features

II. Basic Java Concepts
   1. Java and OOP
   2. Java Structure

III. Installing JDK and Apache NetBeans

# Packages

- In the preceding lessons, the name of each example class was taken from the same name space. This means that a unique name had to be used for each class to avoid name collisions.
- After a while, without some way to manage the name space, you could run out of convenient, descriptive names for individual classes.
- You also need some way to be assured that the name you choose for a class will be reasonably unique and not collide with class names chosen by other programmers.

# Packages

- Java provides a mechanism for partitioning the class name space into more manageable chunks.
- This mechanism is the ***package***. The ***package*** is both a naming and a visibility control mechanism.
- You can define classes inside a package that are not accessible by code outside that package.
- You can also define class members that are exposed only to other members of the same package.

# Defining a Packages

- To create a package is quite easy: simply include a **package** command as the first statement in a Java source file.
- Any classes declared within that file will belong to the specified package.
- The **package** statement defines a name space in which classes are stored. If you omit the **package** statement, the class names are put into the default package, which has no name.
- This is the general form of the **package** statement:

```
package pkg;
```

*pkg* is the name of the package;

# Defining a Packages

- example, the following statement creates a package called **mypackage**:

  *package mypackage;*

- Java uses file system directories to store packages.
- For example, the **.class** files for any classes you declare to be part of **mypackage** must be stored in a directory called **mypackage**.
- The general form of a multileveled package statement is shown here:

*package pkg1[.pkg2[.pkg3]];*

*package a.b.c;*

# Packages and Member Access

- Packages add another dimension to access control. As you will see, Java provides many levels of protection to allow fine-grained control over the visibility of variables and methods within classes, subclasses, and packages.
- Classes and packages are both means of encapsulating and containing the name space and scope of variables and methods.
- Packages act as containers for classes and other subordinate packages.
- Classes act as containers for data and code.

# Packages and Member Access

- Java addresses four categories of visibility for class members:
  - Subclasses in the same package
  - Non-subclasses in the same package
  - Subclasses in different packages
  - Classes that are neither in the same package nor subclasses
- The three access modifiers, **private, public**, and **protected**, provide a variety of ways to produce the many levels of access required by these categories. The following table sums up the interactions.

# Packages and Member Access

| | private | No Modifier | protected | public |
|---|---|---|---|---|
| Same class | Yes | Yes | Yes | Yes |
| Same package subclass | No | Yes | Yes | Yes |
| Same package non-subclass | No | Yes | Yes | Yes |
| Different package subclass | No | No | Yes | Yes |
| Different package non-subclass | No | No | No | Yes |

# Importing packages

- In a Java source file, **import** statements occur immediately following the **package** statement (if it exists) and before any class definitions. This is the general form of the **import** statement:

  *import pkg1 [.pkg2].(classname | *);*

*Example:*

```
package mypackage;

import java.util.Date;

import java.io.*;
```

# Standard Java SE packages

- All of the standard Java SE classes included with Java begin with the name **java**.
- The basic language functions are stored in a package called **java.lang**.
- Normally, you have to import every package or class that you want to use, but since Java is useless without much of the functionality in **java.lang**, it is implicitly imported by the compiler for all programs.

# Standard Java SE packages

| Package Name | Description |
|---|---|
| java.lang | Contains language support classes ( for e.g classes which defines primitive data types, math operations, etc.) . This package is automatically imported. |
| java.io | Contains classes for supporting input / output operations. |
| java.util | Contains utility classes which implement data structures like Linked List, Hash Table, Dictionary, etc and support for Date / Time operations. |
| java.applet | Contains classes for creating Applets. |
| java.awt | Contains classes for implementing the components of graphical user interface ( like buttons, menus, etc. ). |
| java.net | Contains classes for supporting networking operations. |

# Agenda


I. Introduction to Java
1. Java History
2. Java Principles and Features

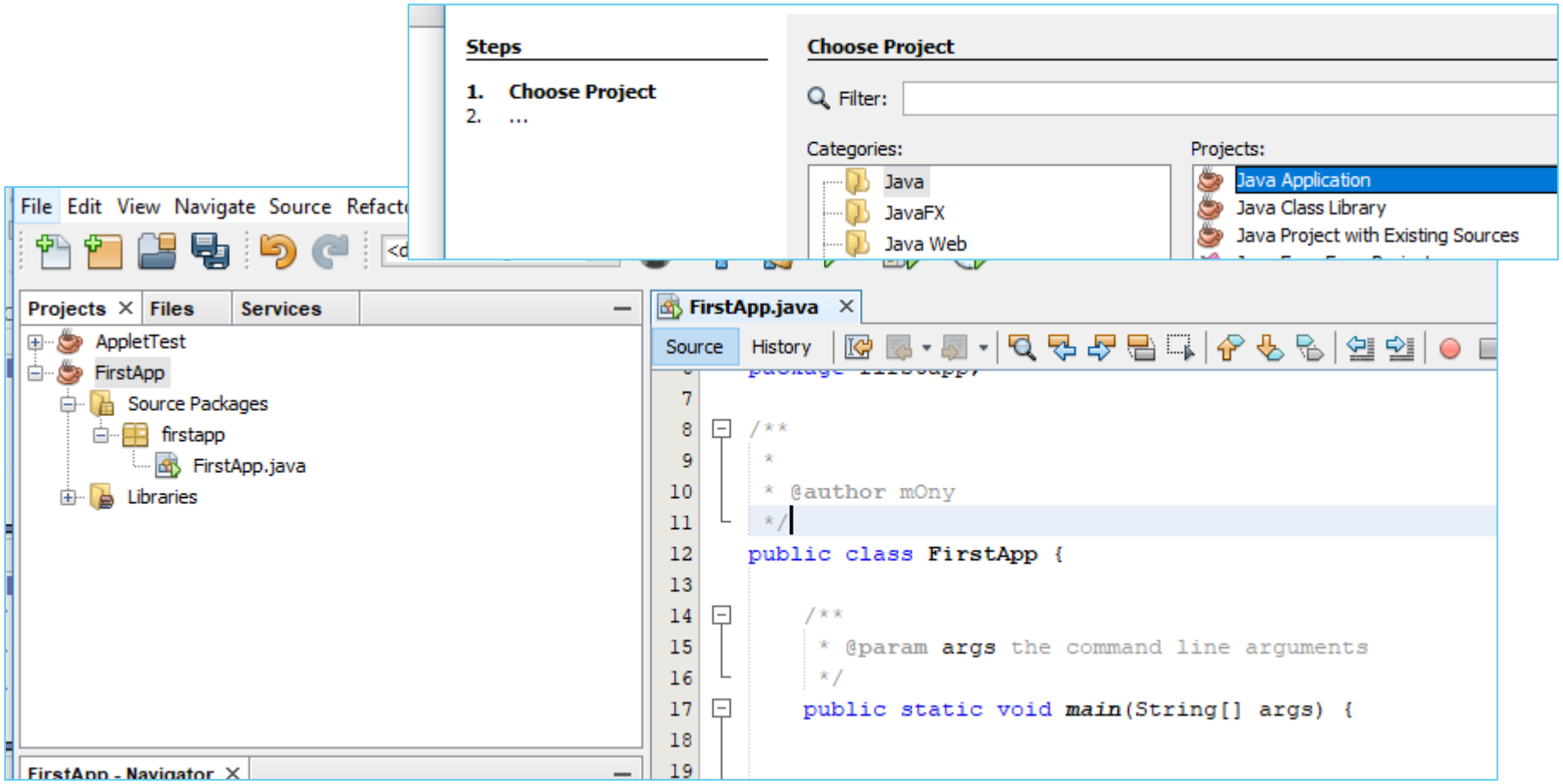
II. Basic Java Concepts
1. Java and OOP
2. Java Structure

III. Installing JDK and Apache NetBeans

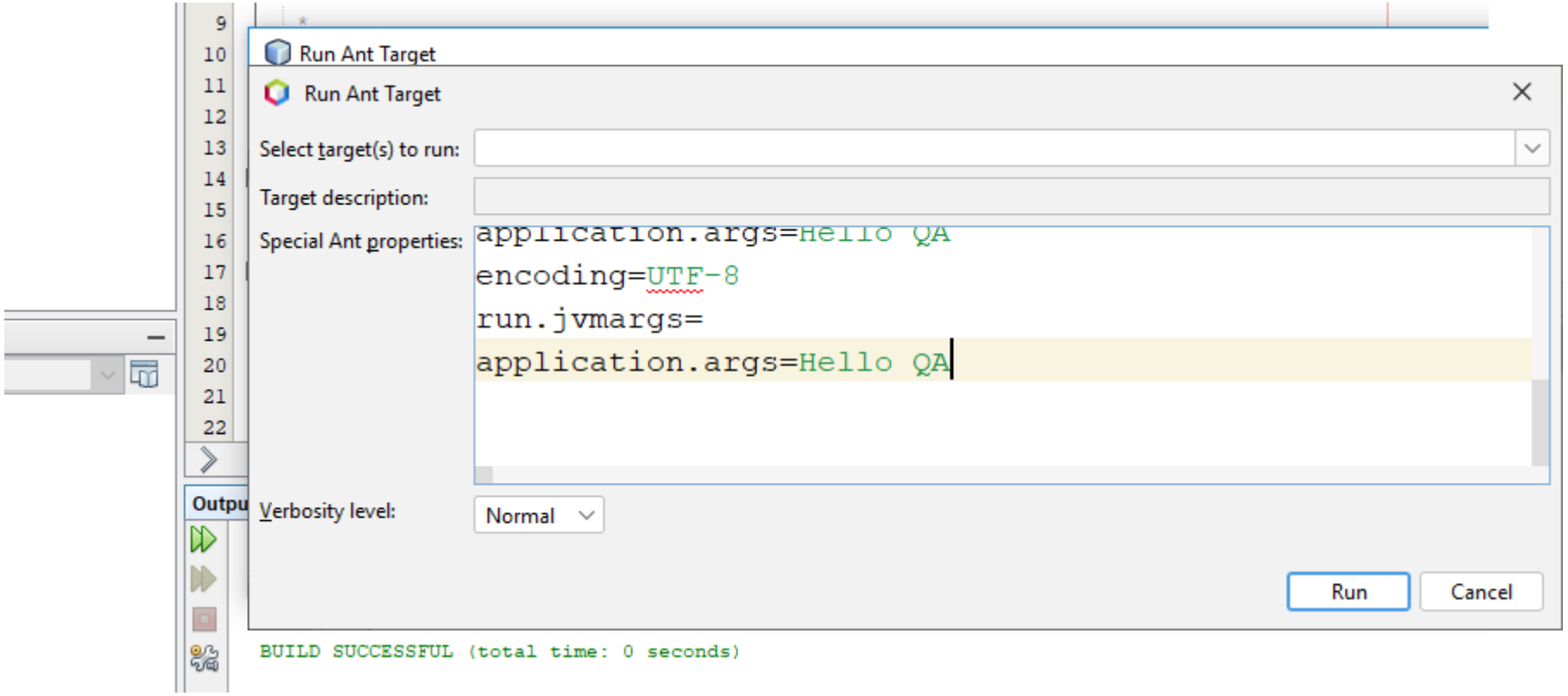
# Installing JDK

- Download the JDK:
  - If you use Solaris, Linux, Windows, or Mac point your browser to http://www.oracle.com/technetwork/java/javase/downloads/index.html to download the JDK.
  - Look for version 8.0 or later, and pick your platform.
  - Download and Install NetBeans or Apache NetBeans.

# @ NetBeans



# @ NetBeans



Java™ Education and Technology Services

# Lab Exercise

# Lab Exercise 1

- Create a simple non-GUI Application that prints out the following text on the command prompt:
  - Hello Java
- **Note:** specify package and create executable jar file.
- Modify the program to print a string that is passed as an argument from the command prompt.